HIGHER ORDER SOFTWARE, INC.
806 Massachusetts Avenue
Post Office Box 531
Cambridge, Massachusetts 02139

TECHNICAL REPORT NO. 28

FURTHER PROGRESS IN KNOWLEDGE REPRESENTATION FOR IMAGE UNDERSTANDING

March 1981

Final Report Prepared For

DEPARTMEMT OF THE NAVY
NAVAL ELECTRONICS SYSTEMS COMMAND
WASHINGTON, DC 20360

HIGHER ORDER SOFTWARE, INC.
806 Massachusetts Avenue
Post Office Box 531
Cambridge, Massachusetts 02139

TECHNICAL REPORT NO. 28

FURTHER PROGRESS IN KNOWLEDGE REPRESENTATION FOR IMAGE UNDERSTANDING

March 1981

Final Report Prepared For

DEPARTMENT OF THE NAVY
NAVAL ELECTRONICS SYSTEMS COMMAND
WASHINGTON, DC 20360

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) 14 HOS-TR-28

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-28✓	2. GOVT ACCESSION NO. AD-A098416	3. RECIPIENT'S CATALOG NUMBER 9 Final
4. TITLE (and Subtitle) 6 FURTHER PROGRESS IN KNOWLEDGE REPRESENTATION FOR IMAGE UNDERSTANDING.		5. ~~TYPE OF REPORT & PERIOD COVERED~~ ~~Interim~~ Report. ~~for period~~ Sept[redacted]80 – Marc[redacted]81
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 S./Cushing L./Vaina 15		8. CONTRACT OR GRANT NUMBER(s) N00039-79-C-0457✓
9. PERFORMING ORGANIZATION NAME AND ADDRESS Higher Order Software, Inc.✓ 806 Massachusetts Avenue / P.O. Box 531 Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Naval Electronics Systems Command Washington, DC 20360		12. ~~REPORT DATE~~ 11 Mar[redacted]81
		13. NUMBER OF PAGES 41
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 44		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release, Unlimited Distribution.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

knowledge-representation, image-processing, areal coordinate systems, HOS, Higher Order Software, data types, specification, fuzzy sets, approximate reasoning, plane decomposition, axiomatics, image understanding, object-views system

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Attention is focused on three problems implicitly raised in the October Interim Report: (1) To provide the basis for and HOS specification of the object-views system (OVS) in terms of which the further development of the system might be expressed; (2) To determine the extent to which a square areal coordinate system can really be used as a basis for image understanding; (3) To determine which formal frameworks for knowledge representation might lend themselves to being useful in dealing with the

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

411 UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

393027

various subclasses of image-derived information that are relveant to the OVS system. Five basic data types are specified in HOS terms, in response to (1), along with some associated operations. Problems in generalizing the modular-vector arithmetic of square coordinate systems beyond the first two levels of aggregates are discussed in connection with (2). Knowledge-representation frameworks that are deemed to be of potential use in the further development of OVS are listed and discussed in response to (3).

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

Work on this project since the appearance of the October Interim Report [1] has focused on three problems implicitly raised in that report:

(1) To provide the basis for an HOS specification of the OVS system in terms of which the further development of the system might be expressed.

(2) To determine the extent to which a square areal coordinate system can really be used as a basis for image understanding.

(3) What formal frameworks for knowledge representation might lend themselves to being useful in dealing with the various sublcasses of image-derived information that are relevant to the OVS system?

In this report we review this work in terms of both the results achieved and the further problems arising from it.

2.0 THE HOSIFICATION OF OVS

OVS is a knowledge-representation system for information derived from visual images, such as radars, and other sources, with the intent of identifying and classifying the real-world objects that serve as the basis for those images. The system is made up of a set of experts, which serve as the information sources, and which issue reports, consisting of information being reported to the data base concerning observed, but as yet not necessarily identified objects. Such information can be of any one of three distinct modes (called "modules" in the earlier report, but "modes" is a less confusing term):

(1) The descriptive mode: information is stored in the form of attribute-value associations; can be summaraized by the code-term "has-a", indicating that a given reported object "has a " specific attribute.

(2) The category mode: information is stored in the form of lists; can be summarized by the code-term "is-a", indicating that a given reported object "is a" member of a specific category of possible objects.

(3) The functional mode: information is a again stored in the form of lists; can be summarized by the code-term "used-for", indicating that a given reported object is "used for" the carrying out of a given function.

The data base itself consists of the sum-total of all reported information, which thus constitutes the knowledge represented and stored in the OVS system.

The full OVS system would require more filling out ~~of~~ the details involved in it, but this gives us its basic underlying components. We thus end up with five essential data types, in terms of which the workings of the system must be expressed:

(1) <u>experts</u>: whatever it is that reports information to the data base;

(2) <u>modes</u>: any one of three kinds of information that an expert can report to the data base;

(3) <u>information</u>: whatever it is that an expert reports to the data base;

(4) <u>knowledge</u>: the data base itself;

(5) <u>reports</u>: whatever form it is in which information gets reported by the experts to the data base.

Constructing HOS specifications for the data types in such a list involves sorting out the interrelationships among the data types themselves and the essential properties and interactions of the members of each data type. These properties are expressed in terms of primitive operations that map into and out of the respective data types in accordance with axioms that serve as constraints on their ultimate implementational behavior. The actual method of constructing such a specification involves a process of successive approximation, in which primitive operations are determined to be needed to express the desired properties and axioms are formulated to express them in terms of these primitive operations. Initial formulations generally reveal simplifications that could be achieved by replacing a primitive operations on one data type with an equivalent one on some other data types or with an HOS operation or structure that removes the purported primitive operation from the data type specification altogether, except perhaps that reference might still be made to it in an axiom, even

```
DATA TYPE:  EXPERT;

PRIMOPS:  ;

AXIOMS:  ;

END EXPERT;
```

Figure 1: Data Type EXPERT

though it is not included in the primitive operation list for that data type. In the present report we give only the final results of this process, in which the simplest and most concise formulation of the data types involved has been achieved; we will mention, in passing, however, alternate choices that might occur to a specifier and that did play a role in earlier approximations, but whose removal or reformulation led to a simplification of the specification.

2.1 Data Type EXPERT

It might appear, at first glance, that KNOWLEDGE should be the data type with which we begin our specification, since this is what we are building with OVS through storing the information contained in reports, but this turns out to be a mistake, because KNOWLEDGE can be characterized only in terms of its interaction with information, which must therefore have been specified already. REPORT, similarly, requires reference to information and experts, while INFORMATION itself has modes as an essential property. Upon reflection and analysis, we realize that, while other data types, except MODE, require ultimate reference to experts in their specification, experts themselves can be anything at all that can issue reports containing information, and this latter characteristic is most naturally included in the specifications of these other data types, rather than in EXPERT itself. It follows that we can view an expert as anything at all and thus provide it with an HOS specification with no primitive operations or axioms, as shown in Figure 1.

2.2 Data Type MODE

Modes are simply the different kinds of information that get reported by experts to the data base, and it does not matter at this layer of specification, what other properties they may have. We have noted, for example, that descriptive-mode information is most naturally stored in the form of attribute-value associations, while category-mode and functional-mode information is most naturally stored in the form of lists, but how information is stored is entirely a matter of implementation, not an essential characteristic of the reported information itself. It follows, therefore, that our HOS specification of data type MODE need only capture the fact that three and only three modes of information exist in OVS, as expressed in the specification in Figure 2. MODE has no primitive operations of its own, because the only thing that we do with modes is associate them with information in reports, and this latter characteristic is most naturally expressed as part of the specification of one of these latter two data types, once MODE itself has already been made available. Since the only characteristic of modes that we have to express is that there are exactly three distinct ones, the only primitive operations we have to use are boolean ones, including equality. The first WHERE statement in Figure 2, therefore, names the three distinct modes, while the axioms guarantee that the three named modes are distinct. The first axiom says that, given any mode at all, that mode is (equal to) one of the three named modes. This tells us that there are, in fact, (at most) three modes. The other three axioms then guarantee that no two modes are the same.

2.3 Data Type INFORMATION

The only significant contraints that we want to put on information are that it constitutes the content of a report and it comes in three modes. The former fact is most naturally (simply, perspicuously, and so on) stated as part of the specification of data type REPORT, once INFORMATION is already available, and the latter can be stated simply by giving INFORMATION a primitive operation that assigns every (piece of) information a mode, as shown in Figure 3. We do not require any axioms on this primitive operation, because what those axioms would say is already included in the specification of

```
DATA TYPE:  MODE;

PRIMOPS:  ;

AXIOMS:  ;

     WHERE Isa, Has, Usedfor ARE CONSTANT MODES;

     WHERE m is a MODE;

Or(Equal(m,Isa),Equal(m,Hasa),Equal(m,Usedfor)) = True;

Equal(Isa,Hasa) = False;

Equal(Isa,Usedfor) = False;

Equal(Hasa,Usedfor) = False;

END MODE;
```

Figure 2: Data Type MODE

```
DATA TYPE:  INFORMATION;

PRIMOPS:  mode = Mode(information);

AXIOMS:  ;

END INFORMATION;
```

Figure 3: Data Type INFORMATION

data type MODE. The fact that there are exactly three kinds of information an expert can report is automatically accounted for by simply providing every information with a mode, because the axioms of Figure 2 already tell us that there are exactly three of these.

2.4 Data Type KNOWLEDGE

Data type KNOWLEDGE is significantly more complex than the three others we have discussed so far, because its simplest formulation involves the use of a non-primitive operation, as well as primitive operations and axioms. KNOWLEDGE is the data base to which experts report information. Each such report updates the data base by adding to it the content of the report, thereby changing the state of the data base. The members of data type KNOWLEDGE are thus the states of the data base, which we can also refer to as states of knowledge. We can account for updating by providing KNOWLEDGE with a primitive operation that inputs a (state of) knowledge and the content of a report and that outputs another (state of) knowledge, but the specification is simplified by having this primitive operation input a knowledge and an information, as shown in Figure 4, leaving the contents of reports for data type REPORT. The axiom in this specification says that, if we update a state of knowledge by a piece of information, then that piece of information is in the resulting state of knowledge, clearly an essential property of an update operation. The Isin operation, itself, however, remains to be characterized, and this could be done by making it a primitive operation with further axioms, but it is simpler to make it a non-primitive operation defined in terms of Update, as shown in Figure 5. What this operation specification tells us is that a piece of information is in the data base, if adding it to the data base would not change the data base, i.e., if adding it to the present state of knowledge leaves the state unchanged. It would be possible to write the equivalent of Figure 5 as an axiom, but this would mask its non-primitive character, primiti-

```
DATA TYPE:  KNOWLEDGE;

PRIMOPS:  knowledge2 = Update(knowledge1,information);

AXIOMS:

       WHERE in IS AN INFORMATION;

       WHERE kn IS A KNOWLEDGE;

Isin(in,Update(kn,in)) = True;

END KNOWLEDGE;
```

Figure 4: Data Type KNOWLEDGE

```
OPERATION:  b = Isin(in,kn);

WHERE b IS A BOOLEAN;

WHERE in IS AN INFORMATION;

WHERE kn, kn' ARE KNOWLEDGES;

   b = Equal(kn',kn) COJOIN kn' = Update(kn,in);

END Isin;
```

Figure 5: Operation Isin

vity being defined as non-decomposability in terms of an HOS control map. Taken together, Figures 4 and 5 characterize both Update and Isin and, thereby, data type KNOWLEDGE.

2.5 Data Type REPORT

The specification of data type REPORT is the most complex of all, containing a number of properties that might otherwise thought to be necessary in the specifications of some other data type. A report consists, in essence, of an author and a content, the author being some expert and the content being some bit of information, in one of the three modes. We could have tried to include this fact in the specifications of expert or information, but this becomes unnecessary, if we say it once and for all in the specification of reports. In effect, the other data type specifications are "incomplete" relative to the full OVS system, because facts about their data types remain unstated, when those specifications are taken by themselves, but the specification of REPORT fills in the missing facts, so that the specification of OVS as a whole is complete. The relation between experts, information, and reports can be formulated, for example, in terms of a primitive operation that assigns every report an expert, its author, and an information, its content, plus an axiom that says that equality of reports amounts to equality of both authors and contents, as shown in Figure 6. The third primitive operation in Figure 6, Before, turns out not to be needed in any axiom on REPORT, but it is needed to formulate a non-primitive operation, After, that does appear in such an axiom. Before is a primitive operation that assigns to a report the state of the data base -- i.e., the state of knowledge -- that existed before the report was made, and After is an operation that assigns to a report the state of knowledge that exists after the report is made. The relation between Before and After is stated in Figure 7, which says that the state of knowledge after a report is obtained by taking the state of knowledge before the report and

```
DATA TYPE:  REPORT;
PRIMOPS:  expert = Author(report);
          information = Content(report);
          knowledge = Before(report);
AXIOMS:
       WHERE r1, r2, r ARE REPORTS;
Equal(r1,r2) = And(Equal(Author(r1), Author(r2),Equal(Content(r1),Content(r2)));
Isin(Content(r),After(r)) = True;
END REPORT;
```

Figure 6: Data Type REPORT

```
OPERATION:  kn = After(r);
WHERE kn,kn' ARE KNOWLEDGES;
WHERE in IS AN INFORMATION;
WHERE r IS A REPORT;
      After:  kn = Update(kn',in) JOIN (kn',in) = f(r);
      f:  kn' = Before(r) COINCLUDE in = Content(r);
END After;
```

Figure 7: Operation After

updating it by the contents of the report. Given this characterization of After, the second axiom in Figure 6 tells us that the content of a report is in the state of knowledge that exists after the report is made, thereby relating Content and After to Isin, and information and reports to knowledge.

2.6 Further Specification Constraints Derivable for the Data Types

Given these data type specifications, many other facts that we would like to be true of the specified data types can be proven as theorems, and, therefore, do not have to be included among the axioms of one or another of the data types. This significantly simplifies the specifications themselves and argues for maximal care in choosing and formulating primitive operations and their associated axioms. For example, we would like to be able to be sure that the content of a report does not change the state of the data base if the information it contains is already in the data base to begin with, i.e., if the report itself is redundant, and, in fact, we can get this result straightforwardly from the axioms we already have:

THEOREM: Isin(Content(r),Before(r)) = Equal(After(r),Before(r))

Proof:

From the definition of Isin (Figure 5),

Isin(in,kn) = Equal(Update(kn,in),kn).

Let in = Content(r)

kn = Before(r), so

Isin(Content(r),Before(r)) = Equal(Update(Before(r),Content(r)),Before(r))).

From the definition of After (Figure 7),

Update(Before(r),Content(r)) = After(r), so

Isin(Content(r),Before(r)) = Equal(After(r),Before(r))

Q.E.D.

In other words, the contents of a report is in the data base, if and only if the state of the data base is the same after the report would be made as it is before the report is made, i.e., reporting information that is already in the data base leaves the state of the data base unchanged. Other essential characteristics that we would want our data types to have can also be proven as theorems in a similar way, so we can conclude that our characterization of them is complete.

2.7 Prospectus

We thus have complete characterizations of the basic data types that we need for our specification of the OVS knowledge-representation system. Further work on the HOSification of OVS would involve the construction of a control map that would capture the actual workings of the system, i.e., the full dynamics through which the members of the data types interact. The construction of such a control map would undoubtedly reveal further data types, operations, and structures that would have to be specified as well, and this would constitute a part of the process of constructing the control map itself. The final result would be a control map representing the dynamic architecture of the OVS system and a set of data type, operation, and structure specifications used within the control map.

The latter items could then be added to the developing HOS library for inclusion in other systems, as the need for them arises.

3.0 AREAL COORDINATE SYSTEMS

Implicit in the earlier report [1] were a number of results concerning the generality of the numerical encodings proposed for each of the illustrated coordinate systems. Zero- and first-level aggregates were shown (based on [2]) in the hexagon system to obey addition tables in which base digits add modulo 7, while the square system with square first-level aggregates were shown to have no suitable addition table. Furthermore, square systems with null or uniform displacement were shown to satisfy addition tables in which base digits add modulo 5 in the case in which aggregates and coding digits "climb" in opposite directions and to have no suitable addition tables in the case in which these "climb" in the same direction. Work in this area since that report has concentrated almost entirely on trying to generalize these results beyond the first level, to second-level aggregates and beyond.

Initial work in this area focused on trying to formulate and prove a theorem to the effect that the given coordinate systems and their numerical encodings are completely general. First-level arithmetic is general, in those systems in which tables exist at all, and an attempt was made to extract an underlying principle from the respective tables which could be extended to higher-level aggregates. All efforts in this direction proved futile and attention was then turned to a direct examination of higher-level arithmetic in order to see whether such a principle could be derived from the higher levels, because of the greater perspective they might provide on the problem of numerical encoding as a whole. This examination led to the discovery that a principle of the desired generality could not exist, because there are counterexamples to the claim it would make on the second level of aggregates.

The kind of problem encountered in this work is illustrated in Figure 8. In this figure are given the zero through third level of

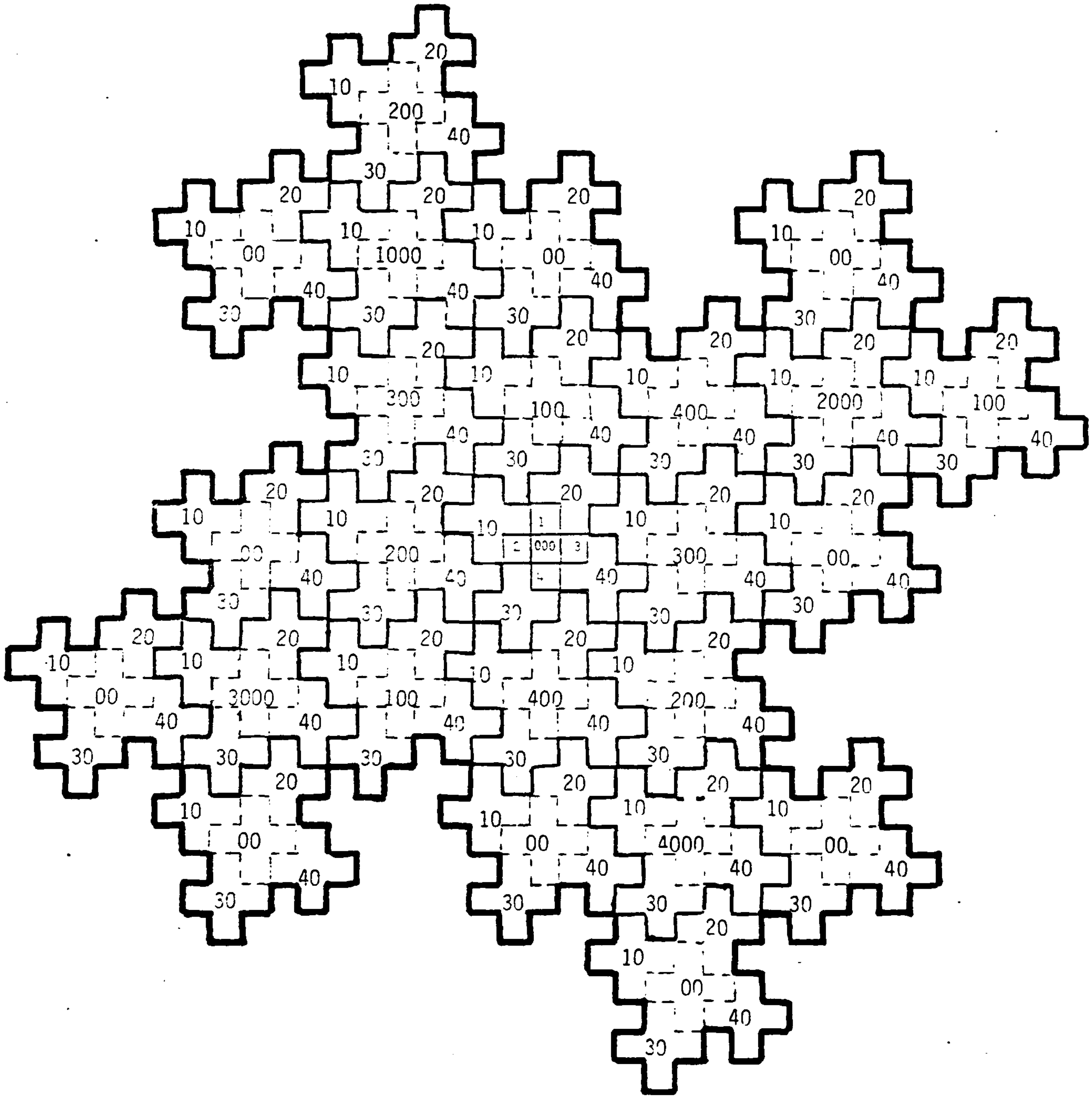

Figure 8: An Additive Square System with Null Displacement

aggregates for a square system with null displacement in which first-level aggregates climb downward and coding digits climb oppositely. As we saw in the earlier report, such a coordinate system should have a completely regular first-level addition table in which base digits add modulo 5, and, in fact, it does, as shown in Figure 9. Given the first-level addition table, the encodings of the second level are forced, and those of the third-level can presumably be derived from them. Perhaps suprisingly, the requirements of the addition force the 20 and 30 first-level aggregates to be reversed from the positions they would be in if their placement followed that of the zero-level aggregates, as is the case in the hexagon system. With this modification, however, first-level addition works perfectly, in accordance with Figure 9, as shown for some examples in Figure 10.

For example, the addition table (plus associativity) tells us that 21 + 21 = 2412, and this is exactly what we find to be the case when we check the locations 21 and 2412 in Figure 8. Similarly, the table tells us that 22 + 43 = 310, and this, too, is verified by checking with Figure 8. Again, according to the table, 43 + 34 = 4212, and this is confirmed by a check with Figure 8.

On the second level of aggregates, the addition also works for most examples, as illustrated in Figure 11. Adding 100 and 300 gives us 2400, according to the table, and also according to Figure 8. The table tells us that 142 + 331 = 2433, and this is confirmed by a look at Figure 8. If we try enough examples on this level, however, we eventually discover that there are also some that do not work, as illustrated in Figure 12. According to the table, for example, 100 plus 100 should be 2200, but Figure 8 puts 100 + 100 in the 1000 aggregate, rather than the 2000 aggregate, which we know to be somewhere else because of the 21 + 21 example in Figure 10.

Such counterexamples to the addition table are the exception on this level, not the rule, but the fact that they exist at all creates

+	0	1	2	3	4
0	0	1	2	3	4
1	1	22	13	24	0
2	2	13	14	0	31
3	3	24	0	41	42
4	4	0	31	42	33

Figure 9: Addition Table for the Square System in Figure 8

(a)

$$\begin{tikzcd}
\begin{array}[b]{c} 21 \\ \underline{21} \\ \end{array} \arrow[r] & \begin{array}[b]{c} 2 \\ 2\cancel{1} \\ \underline{2\cancel{1}} \\ 2 \end{array} \arrow[r] & \begin{array}[b]{c} 14 \\ \cancel{2} \\ \cancel{2}\cancel{1} \\ \underline{2\cancel{1}} \\ 2 \end{array} \arrow[r] & \begin{array}[b]{c} 3 \\ 1\cancel{4} \\ \cancel{2} \\ \cancel{2}\cancel{1} \\ \underline{\cancel{2}\cancel{1}} \\ 12 \end{array} \arrow[r] & \begin{array}[b]{c} \cancel{3} \\ \cancel{1}\cancel{4} \\ \cancel{2} \\ \cancel{2}\cancel{1} \\ \underline{\cancel{2}\cancel{1}} \\ 2412 \end{array}
\end{tikzcd}$$

: 21 + 21 = 2412

(b)

$$\begin{tikzcd}
\begin{array}[b]{c} 22 \\ \underline{43} \\ \end{array} \arrow[r] & \begin{array}[b]{c} 2\cancel{2} \\ \underline{4\cancel{3}} \\ 0 \end{array} \arrow[r] & \begin{array}[b]{c} \cancel{2}\cancel{2} \\ \underline{\cancel{4}\cancel{3}} \\ 310 \end{array}
\end{tikzcd}$$

: 22 + 43 = 310

(c)

$$\begin{tikzcd}
\begin{array}[b]{c} 43 \\ \underline{34} \\ \end{array} \arrow[r] & \begin{array}[b]{c} 4 \\ 4\cancel{3} \\ \underline{3\cancel{4}} \\ 2 \end{array} \arrow[r] & \begin{array}[b]{c} 33 \\ \cancel{4} \\ \cancel{4}\cancel{3} \\ \underline{3\cancel{4}} \\ 2 \end{array} \arrow[r] & \begin{array}[b]{c} 4 \\ 3\cancel{3} \\ \cancel{4} \\ \cancel{4}\cancel{3} \\ \underline{\cancel{3}\cancel{4}} \\ 12 \end{array} \arrow[r] & \begin{array}[b]{c} \cancel{4} \\ \cancel{3}\cancel{3} \\ \cancel{4} \\ \cancel{4}\cancel{3} \\ \underline{\cancel{3}\cancel{4}} \\ 4212 \end{array}
\end{tikzcd}$$

: 43 + 34 = 4212

Figure 10: Examples of Addition on Level 1

(a)

$$\begin{array}{cccc}
100 & 10\cancel{0} & 1\cancel{0}\cancel{0} & \cancel{1}\cancel{0}\cancel{0} \\
\underline{300} & \underline{30\cancel{0}} & \underline{3\cancel{0}\cancel{0}} & \underline{\cancel{3}\cancel{0}\cancel{0}} \\
 & 0 & 00 & 2400
\end{array} \qquad : \; 100 + 300 = 2400$$

(b)

$$\begin{array}{ccccc}
 & & 0 & \cancel{0} & \cancel{0} \\
 & 1 & \cancel{1} & \cancel{1} & \cancel{1} \\
142 & 14\cancel{2} & 1\cancel{4}\cancel{2} & 1\cancel{4}\cancel{2} & \cancel{1}\cancel{4}\cancel{2} \\
\underline{331} & \underline{33\cancel{1}} & \underline{33\cancel{1}} & \underline{3\cancel{3}\cancel{1}} & \underline{\cancel{3}\cancel{3}\cancel{1}} \\
 & 3 & 3 & 33 & 2433
\end{array} \qquad : \; 142 + 331 = 2433$$

Figure 11: Examples of Addition on Level 2

$$\begin{array}{cccc}
100 & 10\cancel{0} & 1\cancel{0}\cancel{0} & \cancel{1}\cancel{0}\cancel{0} \\
\underline{100} & \underline{10\cancel{0}} & \underline{1\cancel{0}\cancel{0}} & \underline{\cancel{1}\cancel{0}\cancel{0}} \\
 & 0 & 00 & 2200
\end{array} \qquad : \; 100 + 100 \overset{?}{=} 2200$$

Figure 12: A Counterexample to Addition on Level 2

problems for the attempt to generalize the numerical encoding of square coordinate systems. Square systems are superior to hexagon systems in the ways stated in the earlier report, but hexagon systems are superior in the ease with which their numerical encodings and arithmetic on level zero can be generalized to higher levels. Square systems could still be made useful for plane decomposition for image processing through the introduction of ad hoc techniques to get around the exceptions, but the optimal solution would be to figure out what the principle is that makes the arithmetic work in those cases in which it does. Future research should involve further attempts to determine this principle and develop its implications for practical use.

4.0 CANDIDATE FORMAL FRAMEWORKS FOR KNOWLEDGE REPRESENTATION

4.1 Introduction

Imagine a robot that has been instructed to collect samples of all the physical objects he perceives. This visual device tells him which are the physical objects what is their size and shape, texture, etc. Once he identifies a physical object, if it is small he collects it (grasps) and puts it into a bag. If the object is big, then he takes a sample of it. If there are any objects that might break, they should be carried. Otherwise, for efficiency, he can just throw them in the bag.

Another robot examines the objects collected in the bag and classifies them by other qualities than size. He throws away those which are not relevant, or those about which the robot is instructed that they are uninteresting because copies of them already exist. He puts the fragile objects separately.

What do these tasks require? They require the ability to perceive and visually represent objects from the environment. Is pure visual representation enough? How does he "put" the object in the bag situated at some distance? This is to ask, how can he "distance" the objects to a new location? In deciding what to do to an object, in order to choose its location, the robot has to know in which actions (throwing, carrying) he can involve the object, More than that, if the object is too big to displace it unsustained, then he should cut a piece of it. Let us suppose that under a big rock something is shining. Thus, the robot whose view was captured by the shine, wants to identify and take a sample of that object. The rock on top of it is too big though, he cannot grasp it, then he decides to push it. And here is the shining little object, thin, and very structured. Thus, since throwing it might break this the robot carries it to the bag.

It would be very useful for the robot to know how to represent knowledge about these objects in the world in relation to the action that can be done upon them safely. This requires more than just 3D and 2½D representation; it requires an action-based representation of the kind we are proposing. But more than that, from this simple example, we infer too that this action-based representation constitutes the basis for higher-level tasks, such as resolving and planning, and even knowledge and belief.

4.2 Knowledge Representation

Representation theory deals with what information we need and how it is represented in the computer. Heuristics is concerned with the structure of the problem-solving algorithms.

The main problem of AI is to find a good answer to the question how a computer can acquire, represent, modify, and make use of knowledge of the world. In this report we review some formal frameworks that show promise for being useful in representing the knowledge acquired from vision. The problems of how this knowledge can be acquired, modified, and made use of should be the topics of further research.

4.3 Inference Mechanisms

The user provides a scalar measure of his reliance on each rule, interpreted as the truth value of the implication underlying the rule. The rule to fire is chosen among the set of rules

$$R_i, \ i = 1,m,$$

by taking into account the qualities of the triggers

$$q_i, \; i = 1,m$$

and the truth values

$$t_i, \; i = 1,m.$$

The choice between R_i and R_j is not straightforward when $q_i > q_j$ and $t_i < t_j$. Then a heuristic is needed to select the rule to fire. Roughly speaking q_i is the adequation of the rule R_i to the current state of the data basis. So, that t_i's can be used to discriminate between the preselected rules.

Inexact Reasoning

Once a rule is fixed, its consequences have to be recorded in the data base. Symbolically a rule is of the form $p_i \longrightarrow C_i$ where p_i is the instantiated trigger and C_i the instantiated consequence.

Two situations are possible: z-values different from 1 are allowed or not to qualify the atoms.

(1) Let z_i be the z-value of the instantiated trigger p_i (i.e., the concatenation of the selected data); z_i is calculated through a conjunction operator (e.g., min) from the z-values of the atoms which are p_i. Two points of view are possible for the evaluation of the truth-value of the premise p_i of the rule to fire R_i: p_i is equal to z_i or p_i is equal to the conjunctive aggregation of z_i and q_i ($\min(z_i,q_i)$ or $z_i \cdot q_i$) if we want to take into account the fact that generally the selected data do not perfectly match the premises.

From p_i and the truth-value t_i of the rule, by means of a detatchment operator $*$ (linked to the multivalent implication implicitly chosen to make $p_i \rightarrow c_i$) a lower bound $b_i * t_i$ of the truth-value c_i of the consequence C_i is computed. All the instantiated atoms, present in the new data built from the consequence, receive C_i as a z-value.

2) Without z-values

If there is no explicit z-value in the system it means $z_i = 1$; then the computed c_i, which is not necessarily 1, is used to validate or invalidate the selection and the consequences of the rule under consideration. If no rule in the preselected subfamily yields a significantly large c_i, the system fails.

<u>Application of Zadeh's Theory of Approximate Reasoning.</u>

Zadeh has developed the following reasoning pattern:

X is A'

if X is A_1, then Y is $B_1 \ldots R_1$

if X is A_n, then Y is $B_n \ldots R_n$

Y is B'

where A_i, A', B_i, B', are fuzzy sets on the universes of discourse U, V, U', V' respectively. A_i, A_i' qualify an attribute of X, B_i, B' qualify an attribute of Y. The membership function of B' is calculated by

$$M_{B'}(V) = \sup \min < M_A(u),$$

$$\max_{i=1,n} (\min(1, 1-M_{A_i}(u) + M_{B_i}(v))), \; u \subset U, \; v \varepsilon V.$$

4.4 Clustering

Suppose S is a set of n objects $o_1, o_2, \ldots o_n$. Important are functions f that assign to every subset of S a number representing the "homogeneity" or "compactness" of the subset. f is 0 for sets with 0 or 1 elements, and positive otherwise. F is the class of such functions. A primitive k-clustering function is any member of F that satisfies the following additional property.

A_k: $f_n\ S_1,\ S_2 \subseteq S,\ f(S_1 \cup S_2) \leq \max\ \{f(S_1),\ f(S_2)\}$

wherever $|S_1 \cap S_2| \geq k$.

The class of all primitive k-clustering functions is denoted by F_k, and obviously, for $k \geq 1$, $F_k \subseteq F_{k+1}$. Whenever $k \geq n-1$, no further restrictions are imposed by assuring A_k holds; i.e., $F_{n-1} = F$. Although properly A_k is of major interest for the initial discussion, a stronger property will be necessary later. Consequently, as a way of introduction and to provide an explicit comparison to A_k, we will call any member of F a k-clustering function if it satisfies F_k.

F_k: $f_n\ S_1, S_2 \subseteq f(S_1 \cup S_2) = \max\ \{f(S_1), f(S_2)\}$

whenever $|S_1 \cap S_2| \geq k$.

Property A_k defines a restriction on the compactness of a set in terms of the compactness of its subsets. A_k as a matter of fact relates directly to the property introduced by Jordine and Gibson as well as to several improtant graph-theoretical concepts relevant to the clustering task.

Primitive k-clustering function:

$\epsilon \geq R$

$P(f,\varepsilon)$ the sets of subsets of S such that

$S_1 \varepsilon P(f,\epsilon)$ iff $F(S_1) \leq \epsilon$

<u>Prop</u>: If $f \varepsilon F_k$, then for any value of ϵ, the distinct elements of P_{max} (f, ϵ) overlap at most k-1 objects. [Note: $P_{max'}(f, \epsilon)$ = those elements of $P(f, \epsilon)$ which are at composition level ϵ.]

Let $S_1, S_2 \varepsilon P_{max}(f, \epsilon)$ where $S_1 \neq S_2$ and assume that S_1 and S_2 overlap more then k-1 elements. Then $|S_1 \cap S_2| \geq k$ and since A_k holds with both $f(S_1) \leq \epsilon$ and $f(S_2) \leq \epsilon$, we know $f(S_1 \cup S_2) \leq \epsilon$, consequently, $S_1 \cup S_2 \varepsilon P(f_1(\epsilon))$. By construction, however $S_1 \subset S_1 \cup S_2$ and $S_2 \subset S_1 \cup S_2$ where both inclusions are proper, and since S_1 and S_2 are maximal in $P(f, \epsilon)$ we have obtained a contradiction. By varying ϵ we obtain a hierarchy of sets of subsets: For $f \varepsilon F$, and $\epsilon \geq \epsilon'$, each element in $P_{max}(f, \epsilon')$ is a subset of at least one element in P_{max} (f, ϵ).

A hierarchy of sets of subsets is defined by f as ϵ varies from 0 to $+ \infty$. The collection $P_{max}(f,0)$ contains only the single-element subsets, and for ϵ sufficiently large, $P_{max}(f, \epsilon)$ contains only S. In a fixed value of ϵ, the subsets develop less then k elements whenever $f \varepsilon F_k$; and thus any function that satisfies A_k may be used to generate a subset hierarchy with the stated overlap property. Clearly for k = 1 a partition hierarchy is obtained, and the constructed sequence of subsets is actually a sequence of partitions.

REMARK: The problem of hierarchical clustering can be characterized by the construction of a hierarchy of sets of subsets; furthermore from my point of view, such a construction would be effectuated through a function $f \varepsilon F$, or if overlap is desired, through $f \varepsilon F_k$.

4.5 Demons

These are procedures which are activated and executed by the successive appearance of certain assertions in the image descriptions or current conceptual database. An essential feature of demons is that the descriptions are continually condensed by the abstraction process: descriptions grow in depth rather than in length.

EXAMPLE: BUILD DEMONS

TYPE	REL	SET OF CONC
1	Between the orientation and trajectory or axis of an object.	Backward, Foreward, Sideways, Around, Over, Clockwise, Counter-clockwise.
2	Between the trajectory of an object and fixed world directions.	Down, Up, North, South, East, West.
3	Changing between objects.	Across, Against, Along, Apart, Around, Away, Away-from, Behind, By From, In, Into.
		Off, Out-off, On, Onto, Over, Through, Together.
4	Indicative of source and target.	Away-from, in-the-direction-of, in, out, toward.
5	Between the path of an object and other (moving) objects.	After, Ahead-of, Along, Apart, Together, With
6	Between an event and previous event.	Back and Forth, To-and-From, Up-and-Down, Back, Through.

4.6 Semantic Nets

Computer programs which exhibit knowlege of the real world when interacting with their users must include powerful mechanisms to represent that knowledge. The greater our demands are for a program to make subtle distinctions, to model its own and other's understanding, and to incorporate new knowledge from its past interactions, the greater are the demands on knowledge representations.

DATA TYPE: The dominant data type of successful symbol manipulation programs has been the expression - a tree of subexpressions with no back pointers. Lisp lists are such expressions, as are production rules and predicate-calculus well formed formulas. For more sophisticated representations, we usually (a) use more complex data structures than the tree, (b) maintain unique structures (c) define a semantics to supplement the criteria of well-formedness. These three techniques are the hallmark of semantic networks (SN). A typical semantic network consists of uniquely represented nodes joined by arbitrary links. This is too general to be of any use to the system builders. Much more restrictive conventions are necessary to constrain the semantics of semantic networks, to make them use more systematic, more meaningful. We need to think of SN with more structure as to allow a well understood semantics. If we can manage to inherit the well understood semantics of first order logic. We are in a good position to make use of the beneficial features of that logic. This opens up the possibility of fast proof-procedure implementation, for example.

The networks we shall deal with are acyclic interconnections of inputs, outputs and gates by means of wires. A network with n inputs and m outputs computes m Boolean function of n arguments.

$b_1(x_1 \ldots, x_n) = z_1$

$\cdot$

$\cdot$

$\cdot$

$b_m(x_1, \ldots, x_n) = z_m$

The gates of a network are the copies of the jobs in a finite basis. With each element in the basis we associate a number called its cost and with each network n we associate a number L(n) called its cost, which is the sum of the cost of the gates it contains. Let b be a function in B^n. L(b) is the minimum of L(B) over all networks that compute b.

SEMANTIC INFORMATION THAT CAN BE USED IN SN:

TYPE SUB-PARTS VISIBILITY MOBILITY LOCATION ORIENTATION SITE	DETERMINE FROM THE IMAGE SEQUENCE

The events are also nodes in the semantic networks.

|| Nodes in Description mode
|| Nodes in Category mode
|| Nodes in Function mode

Each object is potentially the Agent in an event node, or object. A sequence of event nodes forms a history of movement of an object; only the latest node in the sequence is active.

DEFINITION:

SUBJECT: An object which is exhibiting movement.

4.7 Boolean Algebras

A Boolean Algebra is a structure $\langle B, \vee, \wedge, {}^*, 0_B, 1_B \rangle$ consisting of a set B, two binary operations, $\vee, \wedge$ and one unary operation * on B, and two designated elements 0_B, 1_B of B satisfying the following conditions: for any x,y,z b,

$$
\begin{array}{ll}
x \vee y = y \vee x & x \wedge y = y \wedge x \\
x \vee y = x & x \wedge x = x \\
(x \vee y) \wedge y = y & (x \wedge y) \vee y = y \\
(x \vee y) \wedge z = (x \wedge z) \vee (y \wedge z) & (x \wedge y) \vee z = (x \vee z) \wedge (y \vee z) \\
x \vee x^* = 1_B & x \wedge x^* = 0_B
\end{array}
$$

$$0_B \neq 1_B$$

We shall write, for the convenience of the notation. 0 for 0_B and 1 for 1_B, and B for the underlying set of the Boolean objects.

A binary relation $\leq$ on B is defined by

$$x \leq y \Longleftrightarrow x \wedge y = x$$

or equivalently $x \vee y = y$.

The relation $\leq$ is a partial ordering in B, called the natural partial ordering. 0, 1 or the least and the largest element: $x \vee y$ the supremum, and $x \wedge y$ is the infimum, of $\{x,y\}$.

REMARK:

A Boolean algebra B is isomorphic to the algebra of all closed-and-open subsets of a suitable topological space.

A non-empty subset of B is called a subalgebra of B if it is closed under the operations $\vee, \wedge, ^*$ in B.

A homomorphism of one Boolean algebra B into another B' is a map $h: B \to B'$ such that for all $x,y \in B$, $h(x \wedge y) = h(x) \wedge h(y)$, $h(x \vee y) = h(x) \vee h(y)$ and $h(x^*) = h(x)^*$. h must be order-preserving, i.e., $x \leq y \Longrightarrow h(x) \leq h(y)$ for any $x,y \in B$.

An automorphism of B is a one-to-one homomorphism of B onto itself. The collection of all automorphisms of B forms a group under function composition: this group is called the group of automorphisms of B.

A filter in a Boolean algebra B is a non-empty subset F_0 of B such that

(1) $x \in F$, $x \leq y \in B \Longrightarrow y \in F$
(2) $x, y \in F \Longrightarrow x \wedge y \in F$
(3) $0 \notin F$.

A filter F which satisfies, for any x   B;

(4) $x \in F$ or $x^* \in F$

is an ultrafilter.

The language of set theory is a first-order language L with equality which also includes a binary predicate symbol ε

("membership"). The variables of the language, $v_1 \ldots v_n$, range over sets. We also have classes, the term $\{x: \varphi(x)\}$ means "the class of all x such that $\varphi(x)$". We assume that classes satisfy Church's schema

$$\forall y \{ y \varepsilon \{x: \varphi(x)\} \Leftrightarrow \varphi(y)\}$$

4.3 Boolean functions: Complexity of Computing a Function with a Network.

Let $B = \{0,1\}$, where 0 is "false," and 1 stands for "true." B^n is the set of lists of length n over B. The elements of B^n are called points. A Boolean function of n arguments is a function from B^n to B.

Notation

Let us denote with $\mathscr{B}^n$ the set of all such functions. D_n is the number of points in $\mathscr{B}^n$ and $\varphi(n)$ is the number of functions in $\mathscr{B}^n$

$$D_n = 2^n$$

$$\log \varphi(n) = D_n.$$

D_n is the information content of a function in $\mathscr{B}^n$ in the sense that the functions in $\mathscr{B}^n$ can be assigned binary codewords of length D_n. By $L(\mathscr{B}^n)$ we note the complexity of computing an arbitrary function in $\mathscr{B}^n$ with a network. Following Shannon, Luponov showed that

$$L(\mathscr{B}^n) = \rho \frac{2^n}{n} \quad \{1 + O\left(\frac{\log n}{n}\right)\}$$

where ρ is a constant that depends in a known way on the definition of the "network" adopted.

This framework is equivalent to

$$L(\mathscr{B}^n) = \rho \frac{D_n}{\log D_n} \cup \left(\frac{\log \log D_n}{\log D_n}\right)$$

or, obviously, to

$$L(\mathscr{B}^n) = \rho \frac{\log \varphi(n)}{\log \log \varphi(n)} \cup \frac{\log \log \log \varphi(n)}{\log \log \varphi(n)}$$

where $\cup(--)$ deontes a factor of the form $\exp O(--)$.

4.9 Frames

A basic aim of man living in the world is to "make sense of it," to understand where he is, to perceive the situation in which one happens to be, to interpret it, to make decisions about what he needs to do. Thus. whenever one encounters a new situation, he must be able to select from memory a remembered framework and adopt it to fit reality by changing a few details.

Minsky proposes the concept of frame as a basic data structure for representing knowledge about real-world situations. This hypothesis is that one represents in memory a number of stereotyped situations, which are adapted to the real, actual situations. Each frame has attached to it some information about its use and about what can be expected next or what to do if the expectations are not confirmed.

Roughly we can think of a frame as a set of nodes and relations, that form a hierarchical structure. The top level is fixed and contains information which is always done about the supposed situation. The lower levels have slots (terminals) which must be filled with data specific to the real situation. Usually the slots are specific to the conditions its assignments must meet. The simple conditions are markers like the requirement for being a person, or female, etc; sometimes the assignments themselves are simple frames.

A collection of related frames constitutes a frame system. In a frame system the different frames represent different viewpoints of the same situation or object. Several frames may share the same terminals which corresponds to the fact that different views have the same features perceived in different views.

The decision whether a proposed frame is suitable is made by a matching process controlled by one's current goals and by information attached to the frame. Thus, to apply a frame, one has to go through the following steps:

(1) Partial information. Based on the partial evidence or expectation, a frame is evoked.

(2) Current goal. A list of current goals is used to decide which terminals and conditions must be made to match reality.

(3) Value assignment. The terminals that cannot retain their default information need to have new values assigned based on current information.

(4) Transfer of control. If a transformation should happen, then the initial frame would transfer control to the appropriate other frame of the Frame-System.

4.10 Prospectus

These are the knowledge-representation frameworks that seem to us to be potentially useful in dealing with the various subclasses of image-derived information that might become relevant to OVS. Budgetary considerations prevented a thorough comparison and evaluation of the frameworks, but further work could focus on carrying out such an evaluation, with prioritization decisions made as to their actual incorporation into OVS.

REFERENCES

[1] Vaina, L. and S. Cushing, "Foundation of a Knowledge Representation System for Image Understanding", TR-27, Higher Order Software, Inc. Cambridge, MA, October 1980.

[2] Lucas, D., P. Steinfeld, and J.W. Anderson, "System X User's Manual", Martin Marietta (proprietary, used with permission), October 1978.

PRECEDING PAGE BLANK-NOT FILMED